Errors and Exceptions Cheat Sheet

Error Handling

Bug	A **bug** is a flaw in a program's logic. The only solution is to stop execution and find the bug.
Unrecoverable Error	An **unrecoverable error** is a non-bug error where the program must shut down
Recoverable Error	A **recoverable error** is a non-bug related error where a program may continue execution
Result	A **result** will be produced when there is no bug or error, indicates an outcome

Here are the major "handling" paradigms provided by languages.

Roll Your Own	Error Objects	Result Objects
The programmer must "roll their own" handling, like defining enumerated types (success,error) to communicate results.	An "Error" object contains details about an error and nothing else.	A "Result" object contains *either* a valid result (e.g., 5) or the details of an error (e.g., divide by zero).
Older languages: C, Fortran	GoLang	C++, Haskell, Rust, Swift
Optional Objects	**Assertions/Conditions**	**Exceptions and Panics**
An "Optional" object can be used by a function to return a single result that can represent either a valid value or a generic failure condition.	An assertion clause checks whether a required condition is true, and immediately exits the program if it is not.	f() may "throw an exception" which exits f() and all calling functions until the exception is explicitly "caught" and handled by a calling function or the program terminates.
C++, Haskell, Rust, Swift	Most Languages	C++, C#, Java, Python, Swift, ...

1️⃣ Error Objects

Language specific objects that return an explicit error result from a function to its caller, independent of any valid return value.

Languages with error objects provide a built in error class to handle common errors. Error objects are returned along with a function's result as a separate return value

2️⃣ Optionals

Returns a single result that either represents a valid value or generic failure. Only use Optionals if there is an obvious single failure mode.

Think of it like a struct that holds two items: a value and a Boolean indicating whether the value is valid, or an ADT which either contains "nothing" or "something, with a value"

3️⃣ Result Object

Used by a function to return a single result that can represent either a valid value or distinct error. Use when there are multiple failure modes that have to be distinct

You can think of a result as a struct that holds two items: a value and an Error object with details about the nature of the error.

4️⃣ Assertions

A statement inserted into a program that verifies assumptions about your program's state (its variables) that must be true for the correct execution.

Used to verify preconditions, post conditions, and invariants. Good for bugtesting, and when an error would break the code if run

---

Exception Handling

An **exception** is an object with one or more fields which describe an exceptional situation. At minimum, every exception has a way to get a description of the problem. Thrown exception will be directed back at the closest handler in the call stack that can handle its type.

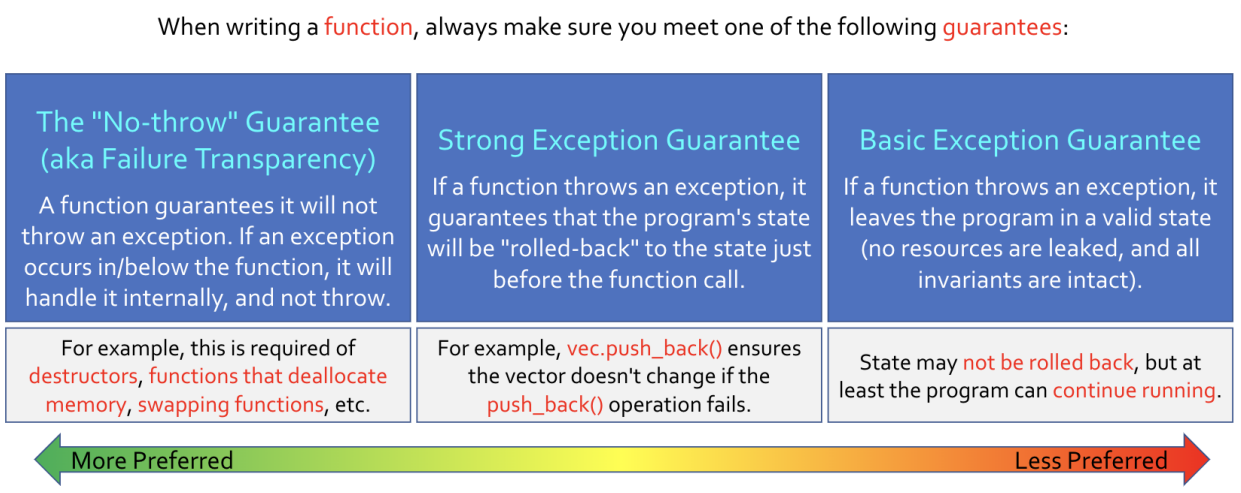
- **Catcher:** has two parts.
  - A block of code that tries to complete one or more operations that might result in an unexpected
  - An exception handler that runs if and only if an error occurs during the tried operations, and deals with the error
- **Thrower:** a function that performs an operation that might result in an error. If an error occurs, the thrower creates an exception object containing details about the error, and “throws” it to the exception handler to deal with.
- **Finally:** Runs regardless of if the try block succeeds or throws

Cons:

- Exceptions are slow and can create bugs. ***__**ONLY USE LANGUAGES WITH AUTOMATIC MEMORY MANAGEMENT**__

Rules for Safe Handling

Make sure at least one of these is guaranteed to be true when writing a function.



Quick Look: What to Use

To check for errors that should never occur if your code is correct (e.g., bugs, unmet preconditions, or violated invariants):

To build unit tests that validate individual classes/functions.

Use assertions

When a function needs to return either a value OR an error for a common, recoverable failure case:

Use an Optional
when there's only one possible failure mode

Use an Error or Result object
when there are multiple failure modes and you need to pass the details of the error to the caller

When a function is unable to fulfill its "contract," AND it's an error that occurs < 1% of the time, AND the failure can be recovered from:

Use exceptions

When there is no reasonable way to recover from an error:

Generate a panic (or uncaught exception)